Robert Burton

Senior Project Write Up

Motion Controlled Graphics Applications

Abstract

This project implements a new control scheme for the OpenGL racing game Crusin Pangea[3] using the motion tracking Leap Motion device[1]. It attempts to increase the engagement with the user by interacting on a more fundamental basis of control. The advantages and drawbacks of this new control scheme are then analyzed based on feedback from a user test group.

1) Introduction

Over the last few decades, technology has continued to become more and more interactive. From the first keyboard, to the mouse, to the touch screen, people have strived to achieve a more vivid connection with the computer. However, all of these methods still rely on physically touching a device. What if a user was able to control a computer by just moving his or her hands? What if they could play a video game without pressing any buttons? Would this make the interactive experience even richer? These are the questions this project focuses on and attempts to answer.

2) Previous/Related Work

The most extensive work in this new field has been accomplished by the developers of Leap Motion. They created a device that, using small cameras and inferred sensors, is able to track the movements of a users hand in 3D space[1]. They have also created an easy to use API for multiple languages including Objective C, Java, C++, and Python. This allows for other developers to continue pushing the frontier of what is possible with the device.

Last quarter in Dr. Kurfess's Knowledge Based System's class, my team worked with the Leap Motion developing gesture recognition software that could be used to navigate webpages. After many weeks of development and testing, we were successfully able to have a user, with minimal to no training, be able to navigate a website using only their hands [2].

Additionally, in another class last quarter, Real-time 3D Computer Graphics, my team worked on creating our own OpenGL racing game "Crusin Pangea." At the end of the quarter, we had a fully functioning keyboard controlled dinosaur racing game, that contained complex environment models, advanced lighting and deferred shading, realistic physics, particle systems, other kart AI, and extensive game logic [3].

These two projects provided the motivation for a motion controlled racing game. Would it be possible? Would it make the game more fun? As the Leap Motion has not been officially released, there are currently no other racing games that utilize this control scheme [4]. Developing and integrating these controls would prove to be both fun and challenging.

3) Development Process

Developing these motion controls for Crusin Pangea took place in three main stages.

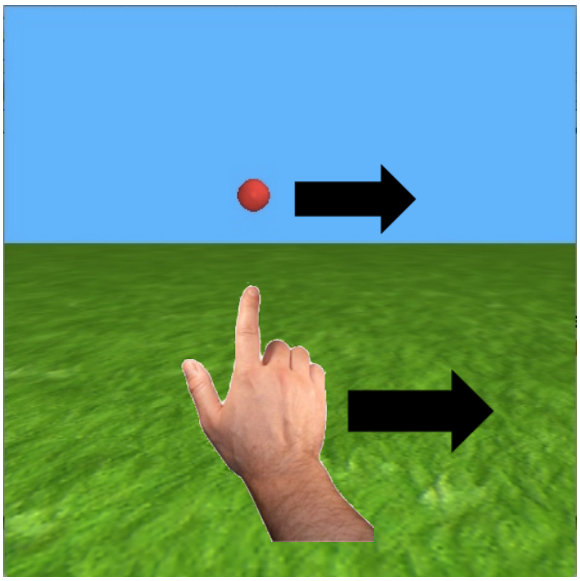
3.1) Integrating Leap Motion with OpenGL to move a sphere

The first step in getting basic functionality was re-familiarizing myself with the Leap Motion development environment. As my last project with Leap Motion had been done entirely in Java, and OpenGL is often done in C++, it was necessary to devote additional time to learning the Leap C++ API. Once accomplished, I created a program that utilized the data from the Leap Motion, and recorded average hand movements over a given period of time. Since the Leap Motion has occasional bad readings, it was necessary to smooth out this data.

The code snippet on the next page shows multiple “frames” being smoothed into one reading. First, all data from the hand is taken in using “hand = frame.hands()[0]”. This assumes one hand, but a multi-hand control scheme could be implemented using an array of these hands. Then, the hand’s finger data are stored in a list of fingers, “fingers”. If the hand has fingers present, or is not only a fist, the hands average finger tip position is then calculated. This is done by looping over all of the fingers, adding each “tipPosition” to a vector “avgPos”. This vector is then divided by the number of fingers. This average position is then added to three variables, avgXPos, avgYPos, and avgZPos, that keep a running sum over the last FRAME_NUM_AVERAGE frames.

Finally, if this is the FRAME_NUM_AVERAGE frame, for example the fifth frame if FRAME_NUM_AVERAGE is qual to five, the running totals avgXPos, avgYPos, and avgZPos are divided by the number of frames. This smoothed data is then sent to OpenGL, and the average positions and frame number are reset.

```
if (!frame.hands().empty()) {
  // Get the first hand
  const Hand hand = frame.hands()[0];

  // Check if the hand has any fingers
  const FingerList fingers = hand.fingers();
  if (!fingers.empty()) {
    // Calculate the hand's average finger tip position
    Vector avgPos;
    for (int i = 0; i < fingers.count(); ++i) {
      avgPos += fingers[i].tipPosition();
    }
    avgPos /= (float)fingers.count();

    avgXPos += avgPos.x;
    avgYPos += avgPos.y;
    avgZPos += avgPos.z;

    frameNum++;

    if(frameNum == FRAME_NUM_AVERAGE)
    {
      avgXPos /= FRAME_NUM_AVERAGE;
      avgYPos /= FRAME_NUM_AVERAGE;
      avgZPos /= FRAME_NUM_AVERAGE;

      //send this data to OpenGL
      sendData(avgXPos, avgYPos, avgZPos);

      frameNum = 0;
      avgXPos = 0;
      avgYPos = 0;
      avgZPos = 0;
    }
  }
}
```



In this figure, as the hand moves to the right, the ball does as well.

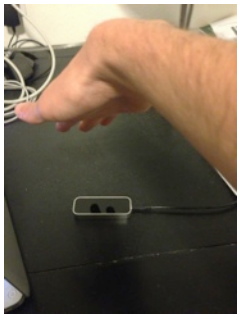
After I was able to compute hand position values, and develop a way to transfer that information, using FILE pointers, all that was left to do was simply draw a sphere at the location specified. The end product looked as pictured above, where the sphere moves through 3D space following your hand or finger.

3.2) Integrate Leap Motion hand controls with simple game

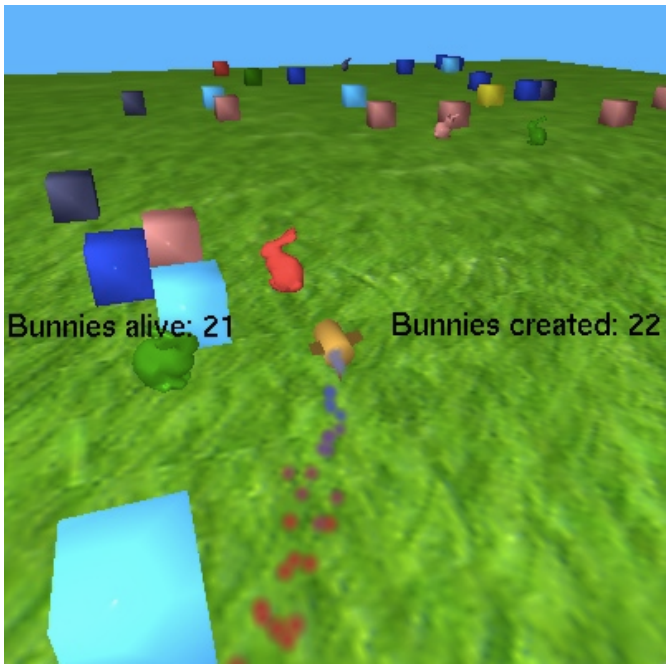
Now that the Leap Motion device and OpenGL were able to communicate, it was necessary to be able to determine the position and rotation of the hand in order to know how to accelerate and turn a vehicle. Calculating this pitch and yaw of the hand utilized much of the same code as one finger, but with a few different substitutions in the API for the Leap Motion. For example, the “roll” of the hand was calculated using multiple different finger locations, and how their Y values were distributed. For example, if there were high Y values on left fingers of the hand and low Y values on the right fingers, this indicated that the hand was tilting clockwise: the control mapping to a right turn.

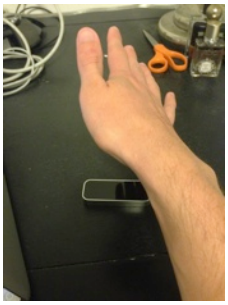
After this was accomplished, additional controls were tested with a simple game involving a rocket chasing bunnies. The goal of the game is to drive the rocket around the world, hitting bunnies while avoiding boxes. The controls of the rocket, instead of being the keyboard were set to the following input controls from the Leap Motion.

Turn left

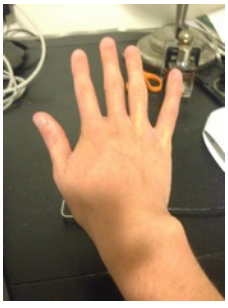


Speed up





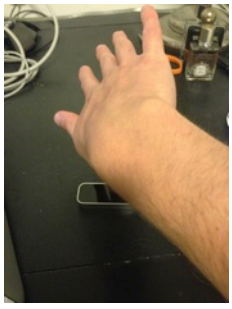
Turn right



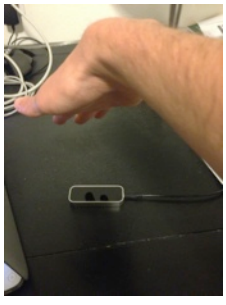
Slow down / reverse

3.3) Integrate Leap Motion hand controls with Crusin Pangea

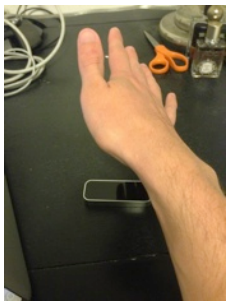
Once the movement controls were fine tuned, it was time to integrate it with the massive project Crusin Pangea: containing over 20,000 lines of code, rendering over 100,000 pieces of geometry per frame, utilizing its own physics engine, and containing over 13 effects from motion blur to particle systems. As this project was so large, one of the hardest parts was discovering exactly where everything needed to go. Since Crusin Pangea was also a team project, my understanding of other team member's code was limited at best. Because the pervious version had only taken discrete key board input (ex. is the forward key pressed or is it not pressed?), the control scheme and even physics code had to be modified to include the possibility of continuous input. For example, if the hand was titled slightly forward, the kart would accelerate less that if the hand was at a shaper angle. This involved changing kart control functions like "accelerateKart(bool)" to "accelerateKart(float amount)." These changed were done for both the "pitch" (for acceleration / deceleration), and "roll" (for turning) of the hand.



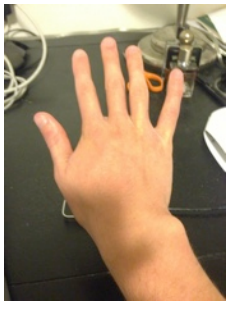
Turn left



Speed up



Turn right



Slow down / reverse

3.4) Performance and Frame-rate Issues

Crusin Pangea renders an extremely large amount of geometry, with a multitude of effects, while extensive game logic runs at the same time. Leap motion utilizes the attached computer's CPU to perform hand tracking hundreds of times per second. Due to how computationally complex both these projects are, the motion controlled version of Crusin Pangea suffered heavily in terms of frame rate. By themselves, on a 2009 MacBook Pro with a 2.4 GHz Intel Core Duo processor, each program would take about 80% of CPU. When run concurrently, neither process had its desired amount of resources. The game was still very much playable, but the experience was drastically reduced due to the choppiness of the images. On average, the game achieved about five frames per second.

Because of this drawback, it was decided the best way to proceed evaluating the Leap Motion controls would be using the much less computationally complex bunny chasing game. It contained much less geometry, minimal effects, and very simple game logic. This allowed for higher frame rate, and more responsiveness from the leap motion, allowing for seemingly instantaneous control. The bunny chasing game was then improved, adding more feed back, visual responsiveness, and additional game play enhancements.

4) Results

To evaluate the overall experience of controlling an OpenGL game with the Leap Motion, ten feedback forms were given out to different individuals with a wide range of ages and interests. The form contained the following five questions:

1. How fun is this motion controlled game?
2. What are the best parts of the game?
3. What aspects could be improved?
4. How do the motion controls enhance or detract from game play?
5. Any other comments/suggestions?

A sample feedback form looked as follows:

Senior Project Feedback Form

Robert Burton

Your name: John Larwood

1. How fun is this motion controlled game?

Not fun 1 2 3 (4) 5 Very fun

2. What are the best parts of the game?

Bunnies flying off into the distance.

3. What aspects could be improved?

- Hook has to angle too far forward to accelerate.
- Turning at slower speeds is not responsive enough

4. How do the motion controls enhance or detract from game play?

The method of control is somewhat tiring, but it definitely makes the game more fun.

5. Any other comments/suggestions?

Make the controller unit a vehicle instead of a rocket if you want the reverse motion to make sense.

Nearly everyone agreed that the motion controls added to the immersion and made the game more fun. One reviewer stated “[The motion controls] make you feel more a part of the game.” Additionally, multiple reviewers favorite part of the game was the ability to control the rocket with your hand. One reviewer, when asked about her favorite part, answered “Using my hand to control the rocket!” A second favorite was the bunnies flying off into the distance when collided with.

While they were generally positively received, one very common complaint about the motion controls was that they were very tiring after extended periods of time. One reviewer stated “Your hand/arm gets tired after playing a lot.” Another user, looking at the silver lining stated “Love it, using my muscles (my arm felt stronger like a workout.)” A solution to this problem could be an arm rest to keep your arm on, or just treating the game as a workout and letting tiredness be another aspect of gameplay.

Additional comments involved making the rocket faster, changing the color of the bunnies to be white, adding a mini-map to see where the remaining bunnies are, adding sound effects, adding an enemy chasing you, adding multi-player to compete for who could hit the most bunnies, and changing the model of the rocket to be a car.

The overall experienced received an average score of 4.0, with 1 being “not fun” and 5 being “very fun.” Analyzing the results, it seems that while the Leap Motion controls are definitely more tiring, they are also more fun as well.

5) Future Work

One area of future work would be improving the rocket bunny game used for demoing the Leap Motion’s controls. There were many suggestions from all the users for aspects that could be improved or added. Some of the most interesting suggestions were enhancing the visuals with trees and lakes, as well as enhancing the game play with an enemy figure, or making the game multiplayer. Since the Leap Motion theoretically has the ability to track two hands at once, it could be interesting to have a split screen mode where two players could play competitively.

Similarly, it could also be interesting to experiment with two hand controls for the game instead of just one. The right hand could be assigned to steering, and the left could be used for power-ups or other controls. This could possibly create too much going on at once, but it would be fun to experiment with.

Lastly, it would be fun to explore creating more OpenGL games with Leap Motion controls. For example, developing a “Temple Run” like game where the user must dodge obstacles using the movements of his or her hand, or a spaceship game that allows full 3D control of the movement of your ship. When put in the right game, these motion controls could add a whole new level of immersive gameplay.

6) References

1. Leap Motion Overview
   https://www.leapmotion.com/product

2. Gesture Recognition with Leap Motion
   https://wiki.csc.calpoly.edu/CSC-481-W13-03/wiki

3. Crusin Pangea Project
   http://users.csc.calpoly.edu/~zwood/teaching/csc476/final13/cjmckee/

4. Leap Motion Ship Date
   http://techcrunch.com/2013/04/25/leap-motion-controller-ship date-delayed-until- july-22-due-to-a-need-for-a-larger-longer-beta-test/